

R and Data Mining: Examples and Case Studies*

Yanchang Zhao
yanchangzhao@gmail.com

April 12, 2011

Contents

1	Introduction	4
1.1	Data Mining	4
1.2	R	4
1.2.1	R Packages and Functions for Data Mining	4
1.3	Datasets	6
1.3.1	Iris Dataset	6
1.3.2	Bodyfat Dataset	7
2	Data Import/Export	8
2.1	Save/Load R Data	8
2.2	Import from and Export to .CSV Files	8
2.3	Import Data from SAS	8
2.4	Import/Export via ODBC	9
2.4.1	Read from Databases	10
2.4.2	Output to and Input from EXCEL Files	10
3	Data Exploration	11
3.1	Have a Look at Data	11
3.2	Explore Individual Variables	12
3.3	Explore Multiple Variables	14
3.4	More Exploration	17
3.5	Save Charts as Files	19
4	Decision Trees	20
4.1	Building Decision Trees with Package <i>party</i>	20
4.2	Building Decision Trees with Package <i>rpart</i>	22
4.3	Random Forest	25
5	Regression	30
5.1	Linear Regression	30
5.2	Logistic Regression	34
5.3	Generalized Linear Regression	34
5.4	Non-linear Regression	36

*The latest version is available at <http://www.rdatamining.com>. See the website also for a *R Reference Card for Data Mining*.

6	Clustering	37
6.1	K-means Clustering	37
6.2	Hierarchical Clustering	38
6.3	Density-based Clustering	39
7	Outlier Detection	43
8	Time Series Analysis	44
8.1	Time Series Decomposition	44
8.2	Time Series Forecast	44
9	Association Rules	46
10	Sequential Patterns	47
11	Text Mining	48
12	Free Online Resources	49
	Bibliography	50

List of Figures

1	Pie Chart	12
2	Histogram	13
3	Density	13
4	Boxplot	14
5	Scatter Plot	15
6	Pairs Plot	16
7	3D Scatter plot	17
8	Level Plot	17
9	Contour	18
10	3D Surface	19
11	Decision Tree	21
12	Decision Tree (Simple Style)	22
13	Decision Tree with <i>rpart</i>	24
14	Prediction Result	25
15	Error Rate of Random Forest	27
16	Variable Importance	28
17	Margin of Predictions	29
18	Australian CPIs in Year 2008 to 2010	30
19	Prediction with Linear Regression Model - 1	33
20	Prediction of CPIs in 2011 with Linear Regression Model	34
21	Prediction with Generalized Linear Regression Model	36
22	Results of K-means Clustering	38
23	Cluster Dendrogram	39
24	Density-based Clustering - I	40
25	Density-based Clustering - II	41
26	Density-based Clustering - III	41
27	Prediction with Clustering Model	42
28	Time Series Decomposition	44
29	Time Series Forecast	45

List of Tables

1 Introduction

This document presents examples and case studies on how to use R for data mining applications.

1.1 Data Mining

Data mining is

The main techniques for data mining are listed below. More detailed introduction can be found in text books on data mining [Han and Kamber, 2000, Hand et al., 2001, Witten and Frank, 2005].

- Clustering:
- Classification:
- Association Rules:
- Sequential Patterns:
- Time Series Analysis:
- Text Mining:

1.2 R

R ¹ [R Development Core Team, 2010a] is a free software environment for statistical computing and graphics. It provides a wide variety of statistical and graphical techniques. R can be extended easily via packages. As on March 11, 2011, there are more than 2800 packages available in the CRAN package repository ². More details about R are available in *An Introduction to R* ³ [Venables et al., 2010] and *R Language Definition* ⁴ [R Development Core Team, 2010c].

R is widely used in academia and research, as well as industrial applications.

1.2.1 R Packages and Functions for Data Mining

A collection of R packages and functions available for data mining are listed below. Some of them are not specially for data mining, but they are included here because they are useful in data mining applications.

1. Clustering

- Packages:
 - *fpc*
 - *cluster*
 - *pvclust*
 - *mclust*
- Partitioning-based clustering: `kmeans`, `pam`, `pamk`, `clara`
- Hierarchical clustering: `hclust`, `pvclust`, `agnes`, `diana`
- Model-based clustering: `mclust`
- Density-based clustering: `dbscan`
- Plotting cluster solutions: `plotcluster`, `plot.hclust`
- Validating cluster solutions: `cluster.stats`

¹<http://www.r-project.org/>

²<http://cran.r-project.org/>

³<http://cran.r-project.org/doc/manuals/R-intro.pdf>

⁴<http://cran.r-project.org/doc/manuals/R-lang.pdf>

2. Classification

- Packages:
 - *rpart*
 - *party*
 - *randomForest*
 - *rpartOrdinal*
 - *tree*
 - *marginTree*
 - *maptree*
 - *survival*
- Decision trees: **rpart**, **ctree**
- Random forest: **cforest**, **randomForest**
- Regression, Logistic regression, Poisson regression: **glm**, **predict**, **residuals**
- Survival analysis: **survfit**, **survdiff**, **coxph**

3. Association Rules and Frequent Itemsets

- Packages:
 - *arules*: supports to mine frequent itemsets, maximal frequent itemsets, closed frequent itemsets and association rules
 - *drm*: regression and association models for repeated categorical data
- APRIORI algorithm, a level-wise, breadth-first algorithm which counts transactions: **apriori**, **drm**
- ECLAT algorithm: employs equivalence classes, depth-first search and set intersection instead of counting: **eclat**

4. Sequential Patterns

- Package: *arulesSequences*
- SPADE algorithm: **cSPADE**

5. Time Series

- Package: *timsac*
- Time series construction: **ts**
- Decomposition: **decomp**, **decompose**, **stl**, **tsr**

6. Statistics

- Package: Base R, *nlme*
- Analysis of Variance: **aov**, **anova**
- Density analysis: **density**
- Statistical test: **t.test**, **prop.test**, **anova**, **aov**
- Linear mixed-effects model fit: **lme**
- Principal components and factor analysis: **princomp**

7. Graphics

- Bar chart: **barplot**
- Pie chart: **pie**

- Scattered plot: `dotchart`
- Histogram: `hist`
- Density: `densityplot`
- Candlestick chart, box plot: `boxplot`
- QQ (quantile-quantile) plot: `qqnorm`, `qqplot`, `qqline`
- Bi-variate plot: `coplot`
- Tree: `rpart`
- Parallel coordinates: `parallel`, `paracoor`, `parcoord`
- Heatmap, contour: `contour`, `filled.contour`
- Other plots: `stripplot`, `sunflowerplot`, `interaction.plot`, `matplot`, `fourfoldplot`, `assocplot`, `mosaicplot`
- Saving graphs: `pdf`, `postscript`, `win.metafile`, `jpeg`, `bmp`, `png`

8. Data Manipulation

- Missing values: `na.omit`
- Standardize variables: `scale`
- Transpose: `t`
- Sampling: `sample`
- Stack: `stack`, `unstack`
- Others: `aggregate`, `merge`, `reshape`

9. Interface to Weka

- *RWeka*: a R/Weka interface enabling to use all Weka functions in R.

1.3 Datasets

The datasets used in this report.

1.3.1 Iris Dataset

Iris dataset consists of 50 samples from each of three classes of iris flowers [Frank and Asuncion, 2010]. One class is linearly separable from the other two, while the latter are not linearly separable from each other. There are five attributes in the dataset:

- sepal length in cm,
- sepal width in cm,
- petal length in cm,
- petal width in cm, and
- class: Iris Setosa, Iris Versicolour, and Iris Virginica.

```
> str(iris)
```

```
'data.frame':      150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

1.3.2 Bodyfat Dataset

Bodyfat is a dataset available in package *mboost*. It has 71 rows, which each row contains information of one person. It contains the following 10 numeric columns.

- **age**: age in years.
- **DEXfat**: body fat measured by DXA, response variable.
- **waistcirc**: waist circumference.
- **hipcirc**: hip circumference.
- **elbowbreadth**: breadth of the elbow.
- **kneebreadth**: breadth of the knee.
- **anthro3a**: sum of logarithm of three anthropometric measurements.
- **anthro3b**: sum of logarithm of three anthropometric measurements.
- **anthro3c**: sum of logarithm of three anthropometric measurements.
- **anthro4**: sum of logarithm of three anthropometric measurements.

The value of **DEXfat** is to be predicted by the other variables.

```
> data("bodyfat", package = "mboost")
> str(bodyfat)
```

```
'data.frame':      71 obs. of  10 variables:
 $ age      : num  57 65 59 58 60 61 56 60 58 62 ...
 $ DEXfat   : num  41.7 43.3 35.4 22.8 36.4 ...
 $ waistcirc : num  100 99.5 96 72 89.5 83.5 81 89 80 79 ...
 $ hipcirc  : num  112 116.5 108.5 96.5 100.5 ...
 $ elbowbreadth: num  7.1 6.5 6.2 6.1 7.1 6.5 6.9 6.2 6.4 7 ...
 $ kneebreadth : num  9.4 8.9 8.9 9.2 10 8.8 8.9 8.5 8.8 8.8 ...
 $ anthro3a   : num  4.42 4.63 4.12 4.03 4.24 3.55 4.14 4.04 3.91 3.66 ...
 $ anthro3b   : num  4.95 5.01 4.74 4.48 4.68 4.06 4.52 4.7 4.32 4.21 ...
 $ anthro3c   : num  4.5 4.48 4.6 3.91 4.15 3.64 4.31 4.47 3.47 3.6 ...
 $ anthro4    : num  6.13 6.37 5.82 5.66 5.91 5.14 5.69 5.7 5.49 5.25 ...
```

2 Data Import/Export

This section shows how to import data into R and how to export R data frames. For more details, please refer to *R Data Import/Export* ⁵ [R Development Core Team, 2010b].

2.1 Save/Load R Data

Data in R can be saved as `.Rdata` files with functions `save`. After that, they can then be loaded into R with `load`.

```
> a <- 1:10
> save(a, file = "E:/Rtmp/dumData.Rdata")
> rm(a)
> load("E:/Rtmp/dumData.Rdata")
> print(a)
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

2.2 Import from and Export to .CSV Files

The example below creates a dataframe `a` and save it as a `.CSV` file with `write.csv`. And then, the dataframe is loaded from file to variable `b` with `read.csv`.

```
> var1 <- 1:5
> var2 <- (1:5) / 10
> var3 <- c("R", "and", "Data Mining", "Examples", "Case Studies")
> a <- data.frame(var1, var2, var3)
> names(a) <- c("VariableInt", "VariableReal", "VariableChar")
> write.csv(a, "E:/Rtmp/dummyData.csv", row.names = FALSE)
> #rm(a)
> b <- read.csv("E:/Rtmp/dummyData.csv")
> print(b)
```

	VariableInt	VariableReal	VariableChar
1	1	0.1	R
2	2	0.2	and
3	3	0.3	Data Mining
4	4	0.4	Examples
5	5	0.5	Case Studies

2.3 Import Data from SAS

Package `foreign` provides function `read.ssd` for importing SAS datasets (`.sas7bdat` files) into R. However, the following points are essential to make importing successful.

- SAS must be available on your computer, and `read.ssd` will call SAS to read SAS datasets and import them into R.
- The file name of a SAS dataset has to be no longer than eight characters. Otherwise, the importing would fail. There is no such a limit when importing from a `.CSV` file.
- During importing, variable names longer than eight characters are truncated to eight characters, which often makes it difficult to know the meanings of variables. One way to get around this issue is to import variable names separately from a `.CSV` file, which keeps full names of variables.

⁵<http://cran.r-project.org/doc/manuals/R-data.pdf>

An empty .CSV file with variable names can be generated with the following method.

1. Create an empty SAS table `dumVariables` from `dumData` as follows.

```
data work.dumVariables;  
    set work.dumData(obs=0);  
run;
```

2. Export table `dumVariables` as a .CSV file.

The example below demonstrates importing data from a SAS dataset. Assume that there is a SAS data file `dumData.sas7bdat` and a .CSV file `dumVariables.csv` in folder “E:/Rtmp”.

```
> library(foreign) # for importing SAS data  
> sashome <- "C:/Program Files/SAS/SAS 9.1"  
> filepath <- "E:/Rtmp"  
> # filename should be no more than 8 characters, without extension  
> fileName <- "dumData"  
> # read data from a SAS dataset  
> a <- read.ssd(file.path(filepath), fileName, sascmd = file.path(sashome, "sas.exe"))  
> print(a)
```

	VARIABLE	VARIABLE2	VARIABLE3
1	1	0.1	R
2	2	0.2	and
3	3	0.3	Data Mining
4	4	0.4	Examples
5	5	0.5	Case Studies

Note that the variable names above are truncated. The full names are imported from a .CSV file with the following code.

```
> variableFileName <- "dumVariables.csv"  
> myNames <- read.csv(paste(filepath, variableFileName, sep="/"))  
> names(a) <- names(myNames)  
> print(a)
```

	VariableInt	VariableReal	VariableChar
1	1	0.1	R
2	2	0.2	and
3	3	0.3	Data Mining
4	4	0.4	Examples
5	5	0.5	Case Studies

Although one can export a SAS dataset to a .CSV file and then import data from it, there are problems when there are special formats in the data, such as a value of “\$100,000” for a numeric variable. In this case, it would be better to import from a .sas7bdat file. However, variable names may need to be imported into R separately.

Another way to import data from a SAS dataset is to use function `read.xport` to read a file in SAS Transport (XPORT) format.

2.4 Import/Export via ODBC

Package *RODBC* provides connection to ODBC databases.

2.4.1 Read from Databases

```
> library(RODBC)
> Conection <- odbcConnect(dsn="servername",uid="userid",pwd="*****")
> Query <- "SELECT * FROM lib.table WHERE ..."
> # or read query from file
> # Query <- readChar("E:/Rtmp/myQuery.sql", nchars=99999)
> myData <- sqlQuery(Conection, Query, errors=TRUE)
> odbcCloseAll()
```

There are also `sqlSave` and `sqlUpdate` for writing or updating a table in an ODBC database.

2.4.2 Output to and Input from EXCEL Files

```
> library(RODBC)
> filename <- "E:/Rtmp/dummyData.xls"
> xlsFile <- odbcConnectExcel(filename, readOnly = FALSE)
> sqlSave(xlsFile, a, rownames = FALSE)
> b <- sqlFetch(xlsFile, "a")
> odbcCloseAll()
```

Note that there is a limit of the number of rows to write to an EXCEL file.

3 Data Exploration

This page shows basic exploration of `iris` data (see Section 1.3.1 for details of iris data).

3.1 Have a Look at Data

Check the dimensionality

```
> dim(iris)
```

```
[1] 150 5
```

Variable names or column names

```
> names(iris)
```

```
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
```

Structure

```
> str(iris)
```

```
'data.frame':      150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

Attributes

```
> attributes(iris)
```

```
$names
```

```
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
```

```
$row.names
```

```
 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
[55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
[73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
[91] 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
[109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
[127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
[145] 145 146 147 148 149 150
```

```
$class
```

```
[1] "data.frame"
```

Get the first 5 rows

```
> iris[1:5,]
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa

```

    Get Sepal.Length of the first 10 rows
> iris[1:10, "Sepal.Length"]
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9

    Same as above
> iris$Sepal.Length[1:10]
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9

```

3.2 Explore Individual Variables

Distribution of every variable

```

> summary(iris)

    Sepal.Length    Sepal.Width    Petal.Length    Petal.Width
Min.   :4.300      Min.   :2.000      Min.   :1.000      Min.   :0.100
1st Qu.:5.100      1st Qu.:2.800      1st Qu.:1.600      1st Qu.:0.300
Median :5.800      Median :3.000      Median :4.350      Median :1.300
Mean   :5.843      Mean   :3.057      Mean   :3.758      Mean   :1.199
3rd Qu.:6.400      3rd Qu.:3.300      3rd Qu.:5.100      3rd Qu.:1.800
Max.   :7.900      Max.   :4.400      Max.   :6.900      Max.   :2.500

    Species
setosa   :50
versicolor:50
virginica :50

```

Frequency

```

> table(iris$Species)

    setosa versicolor  virginica
      50         50         50

```

Pie chart

```

> pie(table(iris$Species))

```

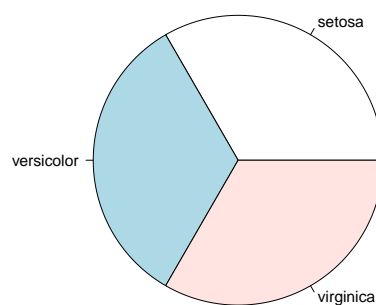


Figure 1: Pie Chart

```
Variance of Sepal.Length  
> var(iris$Sepal.Length)  
[1] 0.6856935
```

```
Histogram  
> hist(iris$Sepal.Length)
```

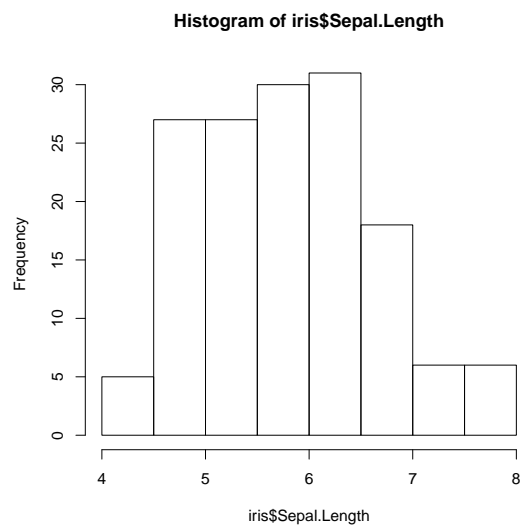


Figure 2: Histogram

```
Density  
> plot(density(iris$Sepal.Length))
```

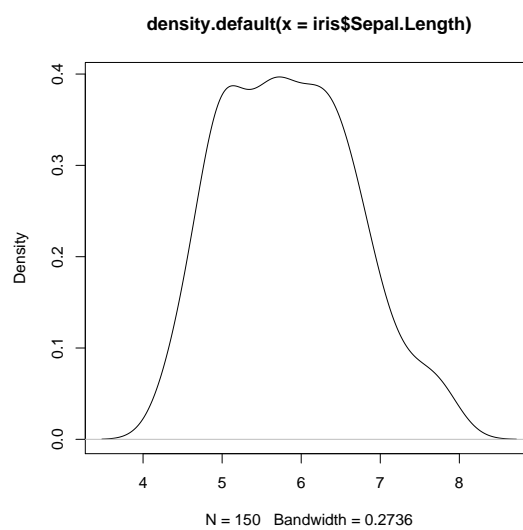


Figure 3: Density

3.3 Explore Multiple Variables

Covariance of two variables

```
> cov(iris$Sepal.Length, iris$Petal.Length)
```

```
[1] 1.274315
```

Correlation of two variables

```
> cor(iris$Sepal.Length, iris$Petal.Length)
```

```
[1] 0.8717538
```

Distribution in subsets

```
> aggregate(Sepal.Length ~ Species, summary, data=iris)
```

	Species	Sepal.Length.Min.	Sepal.Length.1st Qu.	Sepal.Length.Median
1	setosa	4.300	4.800	5.000
2	versicolor	4.900	5.600	5.900
3	virginica	4.900	6.225	6.500

	Sepal.Length.Mean	Sepal.Length.3rd Qu.	Sepal.Length.Max.
1	5.006	5.200	5.800
2	5.936	6.300	7.000
3	6.588	6.900	7.900

Box Plot

```
> boxplot(Sepal.Length~Species, data=iris)
```

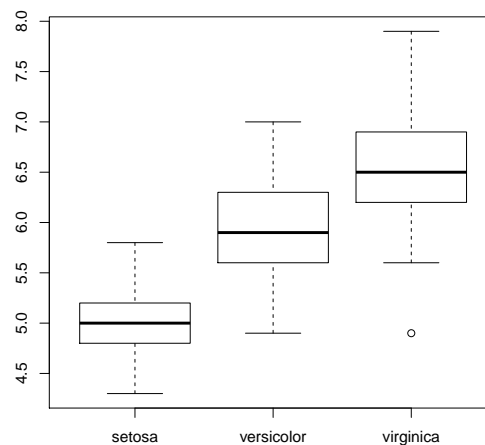


Figure 4: Boxplot

Scatter plot

```
> plot(iris$Sepal.Length, iris$Sepal.Width)
```

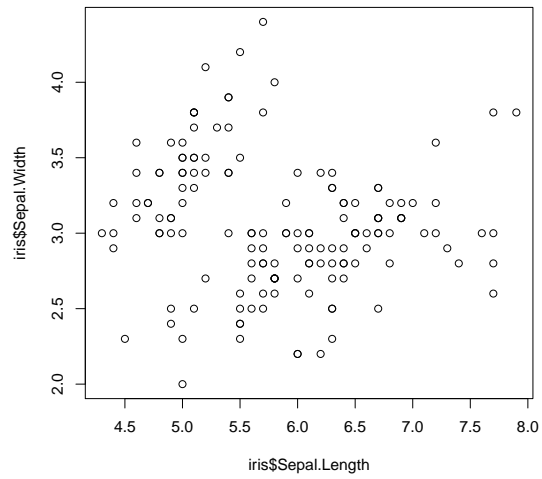


Figure 5: Scatter Plot

Pairs plot

```
> pairs(iris)
```

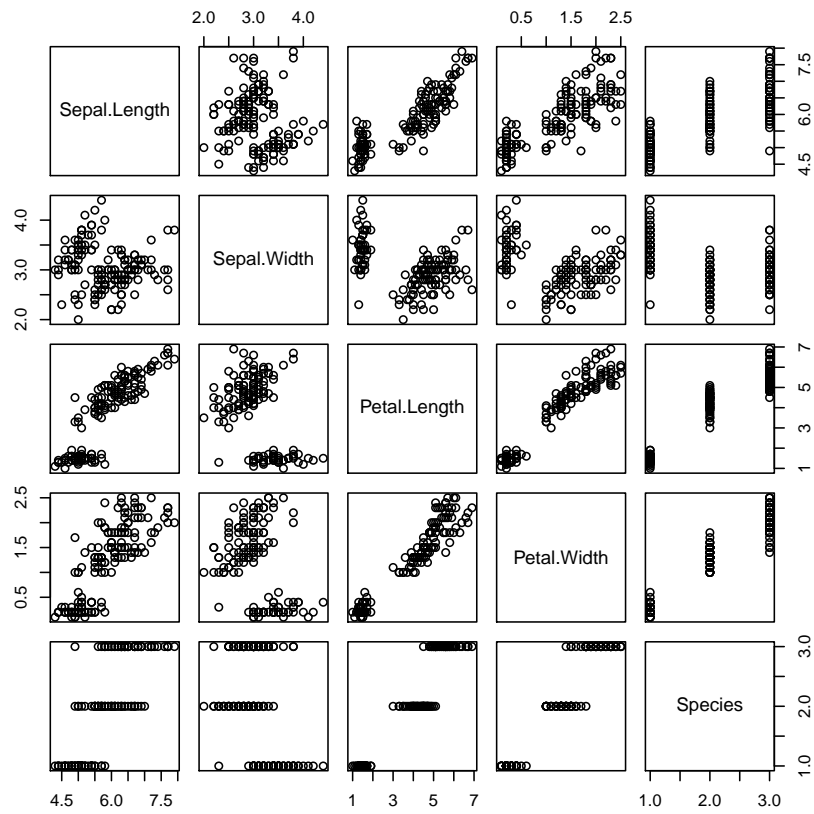


Figure 6: Pairs Plot

3.4 More Exploration

3D Scatter plot

```
> library(scatterplot3d)
> scatterplot3d(iris$Petal.Width, iris$Sepal.Length, iris$Sepal.Width)
```

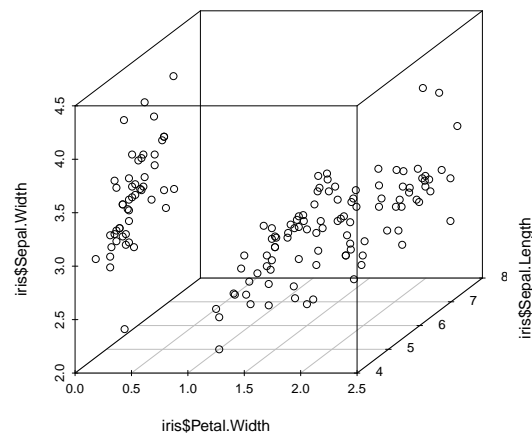


Figure 7: 3D Scatter plot

Level Plot

```
> library(lattice)
> print(levelplot(Petal.Width~Sepal.Length*Sepal.Width, iris, cuts=9, col.regions=grey.colors(10)))
```

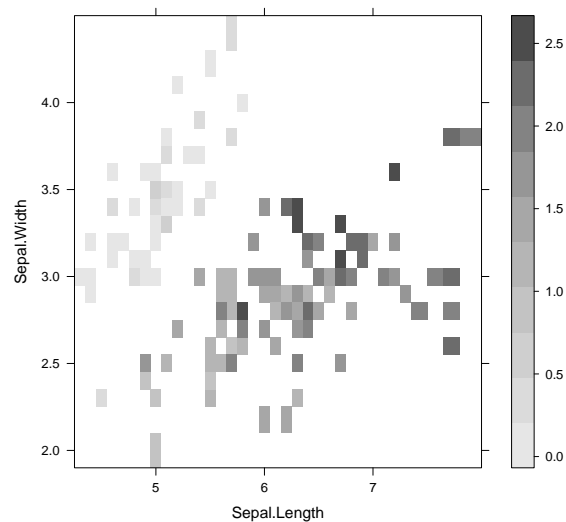


Figure 8: Level Plot

Contour

```
> filled.contour(volcano, color = terrain.colors, asp = 1, plot.axes=contour(volcano, add=T) )
```

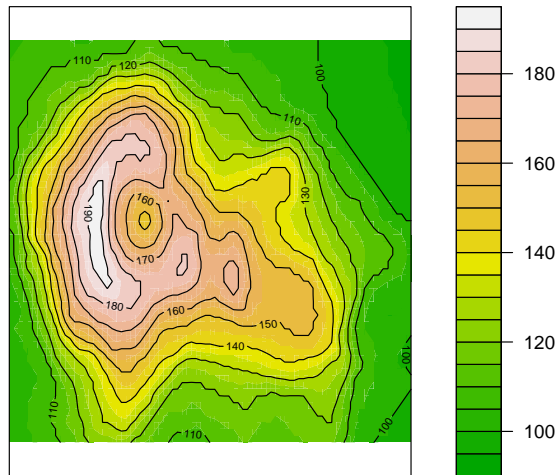


Figure 9: Contour

3D Surface

```
> persp(volcano, theta = 25, phi = 30, expand = 0.5, col = "lightblue")
```

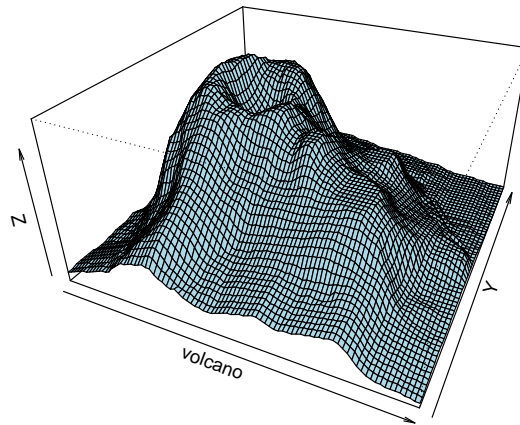


Figure 10: 3D Surface

Interactive 3D Scatter Plot

```
> library(rgl)
> plot3d(iris$Petal.Width, iris$Sepal.Length, iris$Sepal.Width)
```

3.5 Save Charts as Files

Save as a .PDF file

```
> pdf("myPlot.pdf")
> x <- 1:50
> plot(x, log(x))
> graphics.off()
```

Save as a postscript file

```
> postscript("myPlot.ps")
> x <- -20:20
> plot(x, x^2)
> graphics.off()
```

4 Decision Trees

There are a couple of R packages on decision trees, regression trees and random forest, such as *rpart*, *rpartOrdinal*, *randomForest*, *party*, *tree*, *marginTree* and *maptree*.

This section shows how to build prediction models with packages *party*, *rpart* and *randomForest*.

4.1 Building Decision Trees with Package *party*

This section shows how to build a decision tree for *iris* data (see Section 1.3.1 for details of the data) with *ctree* in package *party*. *Sepal.Length*, *Sepal.Width*, *Petal.Length* and *Petal.Width* are used to predict the *Species* of flowers. In the package, function *ctree* builds a decision tree, and *predict* makes prediction for unlabelled data.

The *iris* data is split below into two subsets: training (70%) and testing (30%).

```
> str(iris)

'data.frame':      150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...

> set.seed(1234)
> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
> trainData <- iris[ind==1,]
> testData <- iris[ind==2,]
```

Load package *party*, build a decision tree, and check the prediction.

```
> library(party)
> myFormula <- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
> iris_ctree <- ctree(myFormula, data=trainData)
> # check the prediction
> table(predict(iris_ctree), trainData$Species)
```

	setosa	versicolor	virginica
setosa	40	0	0
versicolor	0	37	3
virginica	0	1	31

Have a look at the built tree.

```
> print(iris_ctree)
```

Conditional inference tree with 4 terminal nodes

Response: Species

Inputs: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width

Number of observations: 112

- 1) Petal.Length <= 1.9; criterion = 1, statistic = 104.643
2)* weights = 40
- 1) Petal.Length > 1.9
3) Petal.Width <= 1.7; criterion = 1, statistic = 48.939
4) Petal.Length <= 4.4; criterion = 0.974, statistic = 7.397

```

5)* weights = 21
4) Petal.Length > 4.4
6)* weights = 19
3) Petal.Width > 1.7
7)* weights = 32

> plot(iris_ctree)

```

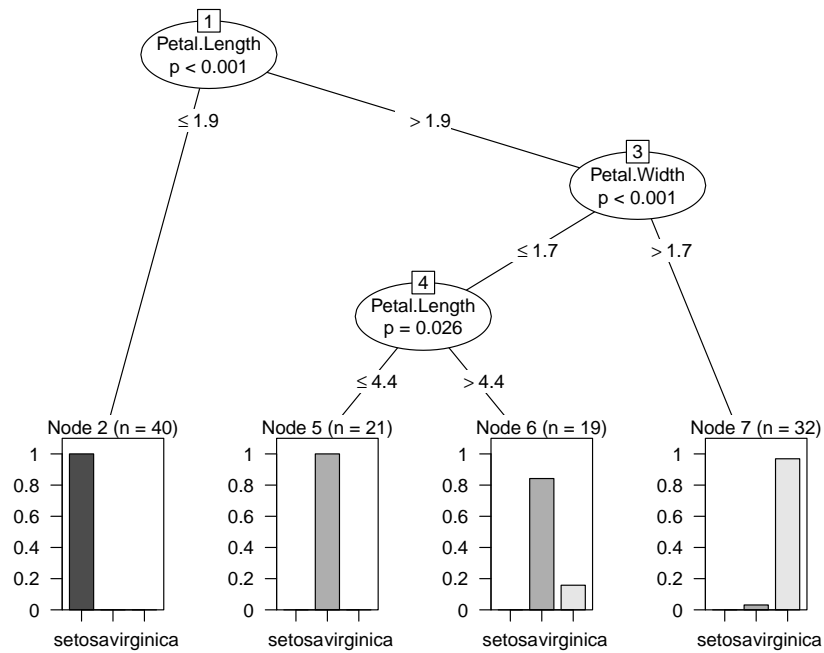


Figure 11: Decision Tree

```
> plot(iris_ctree, type="simple")
```

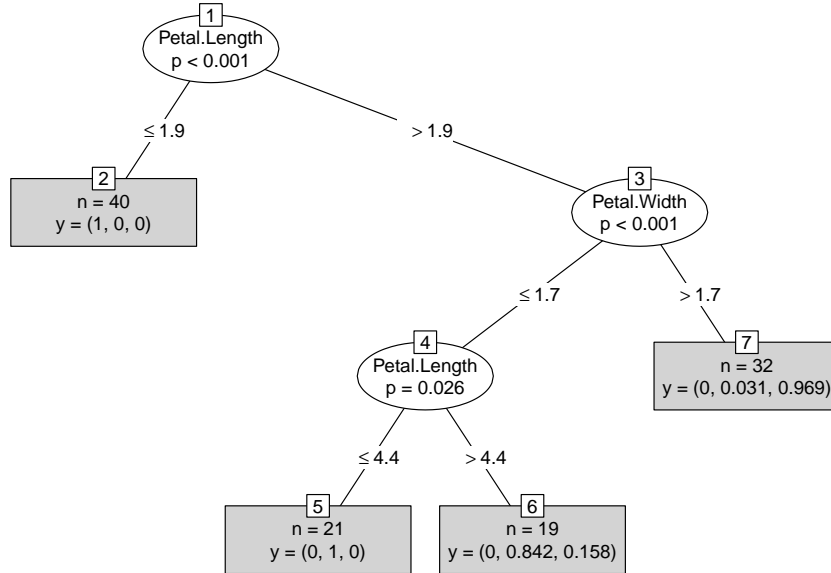


Figure 12: Decision Tree (Simple Style)

Test the built tree with test data.

```
> testPred <- predict(iris_ctree, newdata = testData)
> table(testPred, testData$Species)
```

testPred	setosa	versicolor	virginica
setosa	10	0	0
versicolor	0	12	2
virginica	0	0	14

The current version of `ctree` (i.e. version 0.9-9995) does not handel missing values well. An instance with a missing value may sometimes go to the left sub-tree and sometimes to the right.

Another issue is that, when a variable exists in training data and is fed into `ctree` but does not appear in the built decision tree, the test data must also have that variable to make prediction. Otherwise, a call to `predict` would fail. Moreover, if the value levels of a categorical variable in test data are different from that in train data, it would also fail to make prediction on the test data. One way to get around the above issue is, after building a decision tree, to call `ctree` build a new decision tree with data containing only those variables existing in the first tree, and to explicitly set the levels of categorical variables in test data to the levels of the corresponding variables in train data.

4.2 Building Decision Trees with Package *rpart*

Package *rpart* [Therneau et al., 2010] is used to build a decision tree on `bodyfat` data (see Section 1.3.2 for details of the data). Function `rpart` is used to build a decision tree, and the tree with the minimum prediction error is select. After that, it is applied to makes prediction for unlabelled data with function `predict`.

```
> data("bodyfat", package = "mboost")
> dim(bodyfat)
```

```

[1] 71 10

> attributes(bodyfat)

$names
 [1] "age"          "DEXfat"       "waistcirc"    "hipcirc"      "elbowbreadth"
 [6] "kneebreadth" "anthro3a"     "anthro3b"     "anthro3c"     "anthro4"

$row.names
 [1] "47" "48" "49" "50" "51" "52" "53" "54" "55" "56" "57" "58" "59"
[14] "60" "61" "62" "63" "64" "65" "66" "67" "68" "69" "70" "71" "72"
[27] "73" "74" "75" "76" "77" "78" "79" "80" "81" "82" "83" "84" "85"
[40] "86" "87" "88" "89" "90" "91" "92" "93" "94" "95" "96" "97" "98"
[53] "99" "100" "101" "102" "103" "104" "105" "106" "107" "108" "109" "110" "111"
[66] "112" "113" "114" "115" "116" "117"

$class
[1] "data.frame"

> bodyfat[1:5,]

  age DEXfat waistcirc hipcirc elbowbreadth kneebreadth anthro3a anthro3b anthro3c
47  57 41.68    100.0   112.0         7.1         9.4     4.42     4.95     4.50
48  65 43.29     99.5   116.5         6.5         8.9     4.63     5.01     4.48
49  59 35.41     96.0   108.5         6.2         8.9     4.12     4.74     4.60
50  58 22.79     72.0    96.5         6.1         9.2     4.03     4.48     3.91
51  60 36.42     89.5   100.5         7.1        10.0     4.24     4.68     4.15
  anthro4
47    6.13
48    6.37
49    5.82
50    5.66
51    5.91

> library(rpart)
> myFormula <- DEXfat ~ age + waistcirc + hipcirc + elbowbreadth + kneebreadth
> bodyfat_rpart <- rpart(myFormula, data = bodyfat, control = rpart.control(minsplit = 10))
> attributes(bodyfat_rpart)

$names
 [1] "frame"      "where"      "call"       "terms"      "cptable"    "splits"
 [7] "method"     "parms"      "control"    "functions"  "y"          "ordered"

$class
[1] "rpart"

> print(bodyfat_rpart$cptable)

      CP nsplit  rel error    xerror    xstd
1 0.66289544    0 1.00000000 1.0219736 0.16843699
2 0.09376252    1 0.33710456 0.4035820 0.09175474
3 0.07703606    2 0.24334204 0.4180559 0.08719801
4 0.04507506    3 0.16630598 0.3302274 0.07686651
5 0.01844561    4 0.12123092 0.2520385 0.05796306
6 0.01818982    5 0.10278532 0.2564169 0.05809595
7 0.01000000    6 0.08459549 0.2153285 0.05701589

```

```

> print(bodyfat_rpart)

n= 71

node), split, n, deviance, yval
  * denotes terminal node

1) root 71 8535.98400 30.78282
 2) waistcirc< 88.4 40 1315.35800 22.92375
   4) hipcirc< 96.25 17 285.91370 18.20765
     8) age< 59.5 11 97.00440 15.96000 *
     9) age>=59.5 6 31.45788 22.32833 *
   5) hipcirc>=96.25 23 371.86530 26.40957
     10) waistcirc< 80.75 13 117.60710 24.13077 *
     11) waistcirc>=80.75 10 98.99016 29.37200 *
 3) waistcirc>=88.4 31 1562.16200 40.92355
   6) kneebreadth< 11.15 28 615.52590 39.26036
     12) hipcirc< 109.9 13 136.29600 35.27846 *
     13) hipcirc>=109.9 15 94.46997 42.71133 *
   7) kneebreadth>=11.15 3 146.28030 56.44667 *

> plot(bodyfat_rpart)
> text(bodyfat_rpart, use.n=TRUE)

```

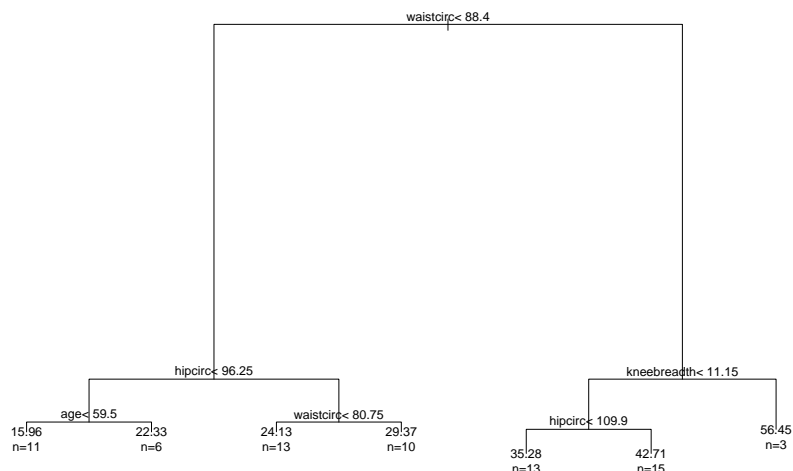


Figure 13: Decision Tree with *rpart*

```

> opt <- which.min(bodyfat_rpart$cptable[, "xerror"])
> cp <- bodyfat_rpart$cptable[opt, "CP"]
> bodyfat_prune <- prune(bodyfat_rpart, cp = cp)
> print(bodyfat_prune)

n= 71

node), split, n, deviance, yval

```

```

* denotes terminal node

1) root 71 8535.98400 30.78282
  2) waistcirc< 88.4 40 1315.35800 22.92375
    4) hipcirc< 96.25 17 285.91370 18.20765
      8) age< 59.5 11 97.00440 15.96000 *
      9) age>=59.5 6 31.45788 22.32833 *
    5) hipcirc>=96.25 23 371.86530 26.40957
      10) waistcirc< 80.75 13 117.60710 24.13077 *
      11) waistcirc>=80.75 10 98.99016 29.37200 *
  3) waistcirc>=88.4 31 1562.16200 40.92355
    6) kneebreadth< 11.15 28 615.52590 39.26036
      12) hipcirc< 109.9 13 136.29600 35.27846 *
      13) hipcirc>=109.9 15 94.46997 42.71133 *
    7) kneebreadth>=11.15 3 146.28030 56.44667 *

> DEXfat_pred <- predict(bodyfat_prune, newdata = bodyfat)

```

The predicted values are compared with real labels.

```

> xlim <- range(bodyfat$DEXfat)
> plot(DEXfat_pred ~ DEXfat, data = bodyfat, xlab = "Observed", ylab = "Predicted", ylim = xlim,
> abline(a = 0, b = 1)

```

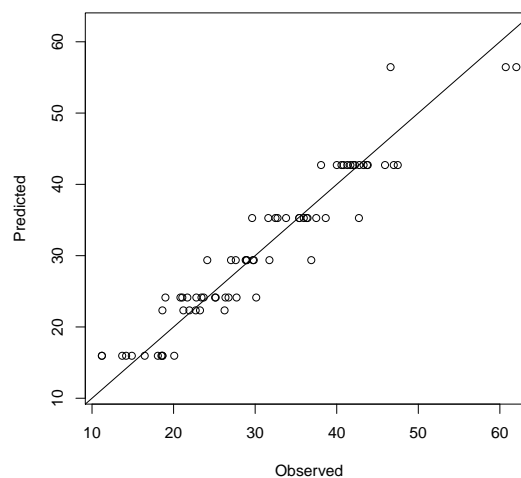


Figure 14: Prediction Result

4.3 Random Forest

Package *randomForest* is used to build a predictive model for *iris* data (see Section 1.3.1 for details of the data). An alternative way is to use function *cforest* from package *randomForest*.

The *iris* data is split below into two subsets: training (70%) and testing (30%).

```

> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
> trainData <- iris[ind==1,]
> testData <- iris[ind==2,]

```

Load *randomForest* and then train a random forest.

```

> library(randomForest)
> rf <- randomForest(Species ~ ., data=trainData, ntree=100, proximity=TRUE)
> table(predict(rf), trainData$Species)

      setosa versicolor virginica
setosa      38         0         0
versicolor   0        33         2
virginica    0         2        28

> print(rf)

Call:
randomForest(formula = Species ~ ., data = trainData, ntree = 100, proximity = TRUE)
      Type of random forest: classification
      Number of trees: 100
No. of variables tried at each split: 2

      OOB estimate of error rate: 3.88%
Confusion matrix:
      setosa versicolor virginica class.error
setosa      38         0         0 0.00000000
versicolor   0        33         2 0.05714286
virginica    0         2        28 0.06666667

> attributes(rf)

$names
[1] "call"          "type"          "predicted"     "err.rate"
[5] "confusion"     "votes"         "oob.times"     "classes"
[9] "importance"    "importanceSD"  "localImportance" "proximity"
[13] "ntree"         "mtry"          "forest"        "y"
[17] "test"          "inbag"         "terms"

$class
[1] "randomForest.formula" "randomForest"

```

Error rates with various number of trees

```
> plot(rf)
```

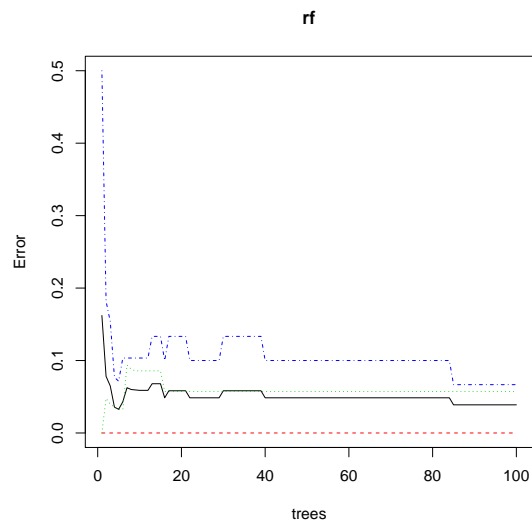


Figure 15: Error Rate of Random Forest

Variable importance.

```
> importance(rf)
```

	MeanDecreaseGini
Sepal.Length	6.653214
Sepal.Width	1.319307
Petal.Length	29.236710
Petal.Width	30.427564

```
> varImpPlot(rf)
```



Figure 16: Variable Importance

Test the built random forest on test data

```
> irisPred <- predict(rf, newdata=testData)
> table(irisPred, testData$Species)
```

irisPred	setosa	versicolor	virginica
setosa	12	0	0
versicolor	0	15	3
virginica	0	0	17

```
> plot(margin(rf, testData$Species))
```

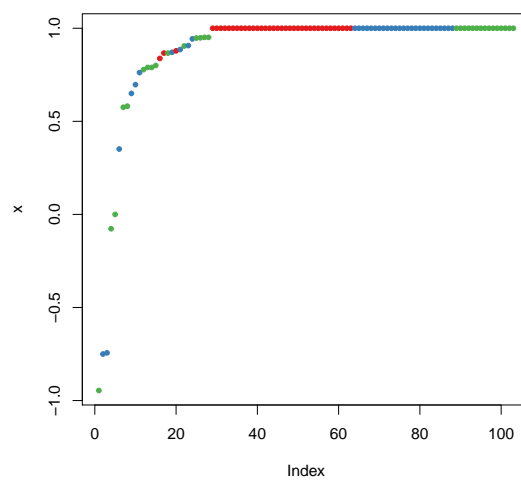


Figure 17: Margin of Predictions

The margin of a data point is as the proportion of votes for the correct class minus maximum proportion of votes for the other classes. Generally speaking, positive margin means correct classification.

5 Regression

Regression is to build a function of *independent variables* (also known as *predictors*) to predict a *dependent variable* (also called *response*). For example, banks assess the risk of home-loan customers based on their age, income, expenses, occupation, number of dependents, total credit limit, etc.

A collection of some helpful R functions for regression analysis is available as a reference card on *R Functions for Regression Analysis* ⁶.

This section will show how to do linear regression with function `lm`, generalized linear regression with `glm`, and non-linear regression with `nls`.

5.1 Linear Regression

Linear regression is to predict response with a linear function of predictors as follows:

$$y = c_0 + c_1x_1 + c_2x_2 + \cdots + c_kx_k,$$

where x_1, x_2, \dots, x_k are predictors and y is the response to predict.

Linear regression is demonstrated below with function `lm` on the Australian CPI (Consumer Price Index) data, which are CPIs in four quarters in every year from 2008 to 2010 ⁷.

```
> year <- rep(2008:2010, each=4)
> quarter <- rep(1:4, 3)
> cpi <- c(162.2, 164.6, 166.5, 166.0, 166.2, 167.0, 168.6, 169.5,
+         171.0, 172.1, 173.3, 174.0)
> plot(cpi, xaxt="n", ylab="CPI", xlab="")
> # draw x-axis
> axis(1, labels=paste(year,quarter,sep="Q"), at=1:12, las=3)
```

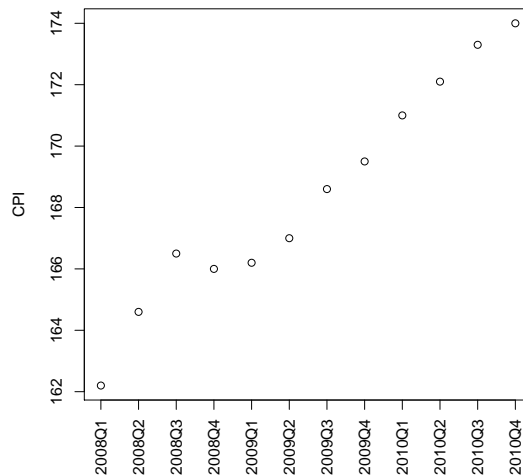


Figure 18: Australian CPIs in Year 2008 to 2010

We check the correlation between CPI and the other variables, `year` and `quarter`.

```
> cor(year, cpi)
```

⁶<http://cran.r-project.org/doc/contrib/Ricci-refcard-regression.pdf>

⁷From Australian Bureau of Statistics <<http://www.abs.gov.au>>

```
[1] 0.9096316
```

```
> cor(quarter, cpi)
```

```
[1] 0.3738028
```

Then a linear regression model is built on the above data, using `year` and `quarter` as predictors and `CPI` as response.

```
> fit <- lm(cpi ~ year + quarter)
> fit
```

Call:

```
lm(formula = cpi ~ year + quarter)
```

Coefficients:

(Intercept)	year	quarter
-7644.487	3.887	1.167

With the above linear model, CPI is calculated as

$$\text{cpi} = c_0 + c_1 * \text{year} + c_2 * \text{quarter},$$

where c_0 , c_1 and c_2 are coefficients from model `fit`. Therefore, the CPIs in 2011 can be get as follows. A simpler way for this is using function `predict`, which will be demonstrated at the end of this subsection.

```
> cpi2011 <- fit$coefficients[[1]] + fit$coefficients[[2]]*2011 + fit$coefficients[[3]]*(1:4)
```

More details of the model:

```
> attributes(fit)
```

\$names

[1] "coefficients"	"residuals"	"effects"	"rank"
[5] "fitted.values"	"assign"	"qr"	"df.residual"
[9] "xlevels"	"call"	"terms"	"model"

\$class

```
[1] "lm"
```

```
> fit$coefficients
```

(Intercept)	year	quarter
-7644.487500	3.887500	1.166667

The differences between observed values and fitted values are

```
> residuals(fit)
```

1	2	3	4	5	6
-0.57916667	0.65416667	1.38750000	-0.27916667	-0.46666667	-0.83333333
7	8	9	10	11	12
-0.40000000	-0.66666667	0.44583333	0.37916667	0.41250000	-0.05416667

```
> summary(fit)
```

```

Call:
lm(formula = cpi ~ year + quarter)

Residuals:
    Min       1Q   Median       3Q      Max
-0.8333 -0.4948 -0.1667  0.4208  1.3875

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -7644.4875   518.6543  -14.739 1.31e-07 ***
year          3.8875     0.2582   15.058 1.09e-07 ***
quarter       1.1667     0.1885    6.188 0.000161 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7302 on 9 degrees of freedom
Multiple R-squared:  0.9672,    Adjusted R-squared:  0.9599
F-statistic: 132.5 on 2 and 9 DF,  p-value: 2.108e-07

```

```
> plot(fit)
```

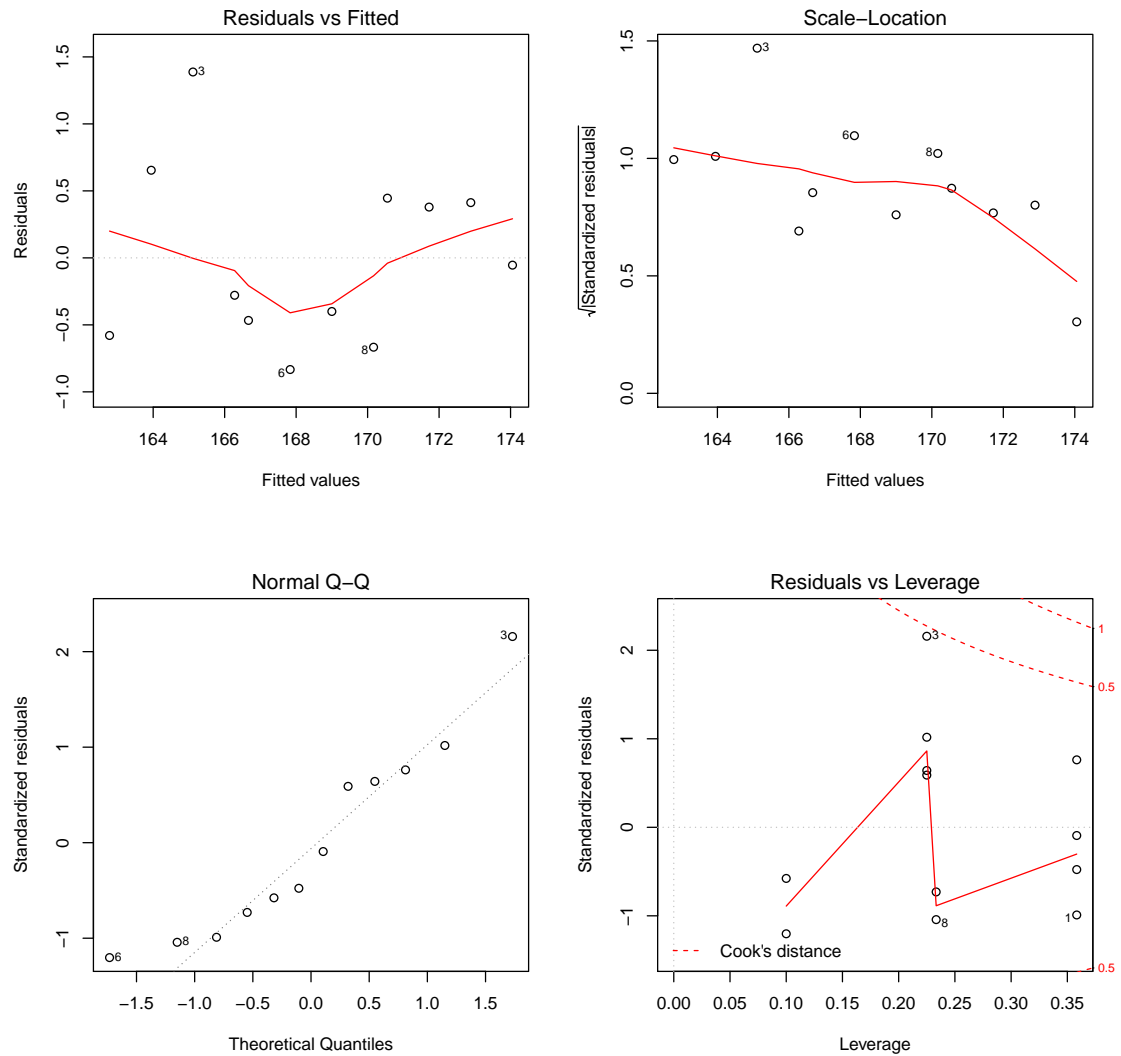


Figure 19: Prediction with Linear Regression Model - 1

With the model, the CPIs in year 2011 can be predicted as follows, and the predicted values are shown as red triangles in Figure 20.

```

> data2011 <- data.frame(year=2011, quarter=1:4)
> cpi2011 <- predict(fit, newdata=data2011)
> style <- c(rep(1,12), rep(2,4))
> plot(c(cpi, cpi2011), xaxt="n", ylab="CPI", xlab="", pch=style, col=style)
> axis(1, at=1:16, las=3,
+      labels=c(paste(year,quarter,sep="Q"), "2011Q1", "2011Q2", "2011Q3", "2011Q4"))

```

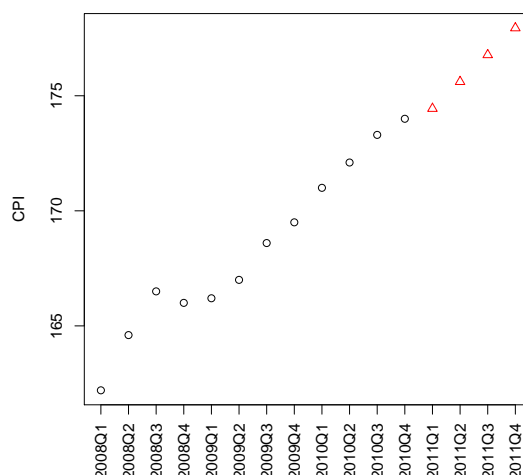


Figure 20: Prediction of CPIs in 2011 with Linear Regression Model

5.2 Logistic Regression

Logistic regression is used to predict the probability of occurrence of an event by fitting data to a logistic curve. A logistic regression model is built as the following equation:

$$\text{logit}(y) = c_0 + c_1x_1 + c_2x_2 + \cdots + c_kx_k,$$

where x_1, x_2, \dots, x_k are predictors, y is a response to predict, and $\text{logit}(y) = \ln(\frac{y}{1-y})$. The above equation can also be written as

$$y = \frac{1}{1 + e^{-(c_0 + c_1x_1 + c_2x_2 + \cdots + c_kx_k)}}.$$

Logistic regression can be built with function `glm` by setting `family` to `binomial(link="logit")`. Detailed introductions on logistic regression can be found at the following links.

- R Data Analysis Examples - Logit Regression
<http://www.ats.ucla.edu/stat/r/dae/logit.htm>
- Logistic Regression (with R)
<http://nlp.stanford.edu/~manning/courses/ling289/logistic.pdf>

5.3 Generalized Linear Regression

The generalized linear model (GLM) generalizes linear regression by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value. It unifies various other statistical models, including linear regression, logistic regression and Poisson regression. Function

`glm` is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

A generalized linear model is built below with `glm` on `bodyfat` data (see Section 1.3.2 for details of the data).

```
> data("bodyfat", package = "mboost")
> myFormula <- DEXfat ~ age + waistcirc + hipcirc + elbowbreadth + kneebreadth
> bodyfat.glm <- glm(myFormula, family = gaussian("log"), data = bodyfat)
> summary(bodyfat.glm)
```

Call:

```
glm(formula = myFormula, family = gaussian("log"), data = bodyfat)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-11.5688	-3.0065	0.1266	2.8310	10.0966

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.734293	0.308949	2.377	0.02042 *
age	0.002129	0.001446	1.473	0.14560
waistcirc	0.010489	0.002479	4.231	7.44e-05 ***
hipcirc	0.009702	0.003231	3.003	0.00379 **
elbowbreadth	0.002355	0.045686	0.052	0.95905
kneebreadth	0.063188	0.028193	2.241	0.02843 *

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 20.31433)

Null deviance: 8536.0 on 70 degrees of freedom
 Residual deviance: 1320.4 on 65 degrees of freedom
 AIC: 423.02

Number of Fisher Scoring iterations: 5

```
> pred <- predict(bodyfat.glm, type = "response")
```

In the code above, `type` indicates the type of prediction required. The default is on the scale of the linear predictors, and the alternative `"response"` is on the scale of the response variable.

```
> plot(bodyfat$DEXfat, pred, xlab="Observed Values", ylab="Predicted Values")
```

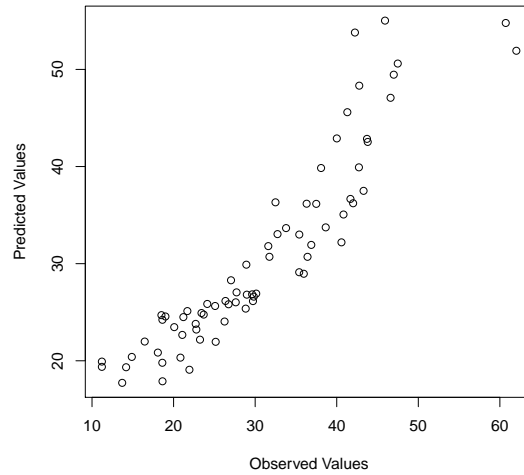


Figure 21: Prediction with Generalized Linear Regression Model

In the above code, if `family = gaussian("identity")` is used, the resulted model would be similar to linear regression. One can also make it a logistic regression by setting `family` to `binomial("logit")`.

5.4 Non-linear Regression

While linear regression is to find the line that comes closest to data, non-linear regression is to fit a curve through data. Function `nls` provides nonlinear regression. More details on non-linear regression can be found at

- A Complete Guide to Nonlinear Regression
<http://www.curvefit.com/>.

6 Clustering

6.1 K-means Clustering

This page demonstrates k-means clustering of `iris` data (see Section 1.3.1 for details of the data)..

```
> newiris <- iris
> newiris$Species <- NULL
```

Apply `kmeans` to `newiris`, and store the clustering result in `kc`. The cluster number is set to 3.

```
> (kc <- kmeans(newiris, 3))
```

K-means clustering with 3 clusters of sizes 38, 50, 62

Cluster means:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	6.850000	3.073684	5.742105	2.071053
2	5.006000	3.428000	1.462000	0.246000
3	5.901613	2.748387	4.393548	1.433871

Clustering vector:

```
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[38] 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[75] 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1
[112] 1 1 3 3 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 3 1 1 1 1 3 1 1 1 3 1
[149] 1 3
```

Within cluster sum of squares by cluster:

```
[1] 23.87947 15.15100 39.82097
(between_SS / total_SS = 88.4 %)
```

Available components:

```
[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"
```

Compare the `Species` label with the clustering result

```
> table(iris$Species, kc$cluster)
```

	1	2	3
setosa	0	50	0
versicolor	2	0	48
virginica	36	0	14

The above result shows that cluster “setosa” can be easily separated from the other clusters, and that clusters “versicolor” and “virginica” are to a small degree overlapped with each other.

Plot the clusters and their centres. Note that there are four dimensions in the data and that only the first two dimensions are used to draw the plot below. Some black points close to the green centre (asterisk) are actually closer to the black centre in the four dimensional space. Note that the results of k-means clustering may vary from run to run, due to random selection of initial cluster centres.

```
> plot(newiris[c("Sepal.Length", "Sepal.Width")], col = kc$cluster)
> points(kc$centers[,c("Sepal.Length", "Sepal.Width")], col = 1:3, pch = 8, cex=2)
```

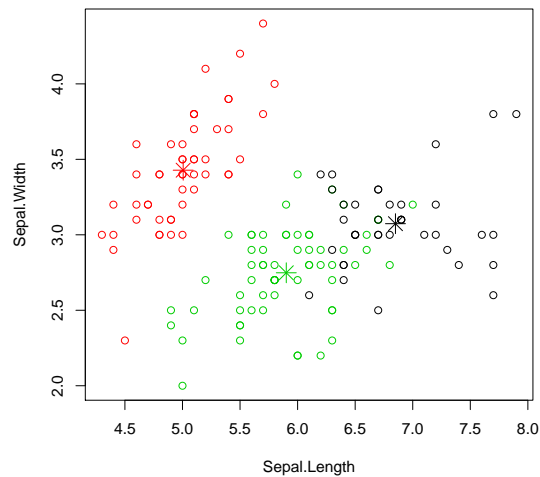


Figure 22: Results of K-means Clustering

6.2 Hierarchical Clustering

This page demonstrates a hierarchical clustering with `hclust` on `iris` data (see Section 1.3.1 for details of the data).

Draw a sample of 40 records from `iris` data, and remove variable `Species`

```
> idx <- sample(1:dim(iris)[1], 40)
> irisSample <- iris[idx,]
> irisSample$Species <- NULL
```

Hierarchical clustering

```
> hc <- hclust(dist(irisSample), method="ave")
```

```
> plot(hc, hang = -1, labels=iris$Species[idx])
```

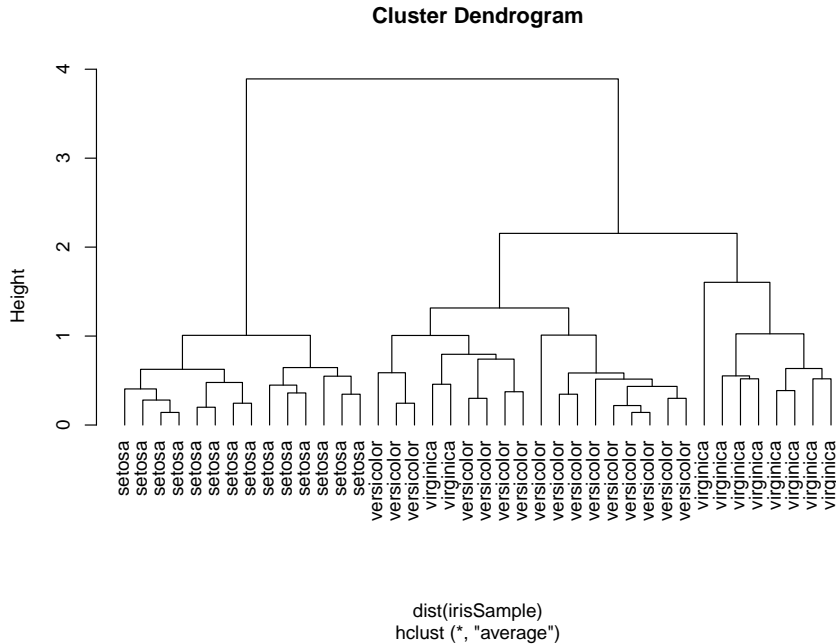


Figure 23: Cluster Dendrogram

Similar to the above clustering of k-means, Figure 23 also shows that cluster “setosa” can be easily separated from the other two clusters, and that clusters “versicolor” and “virginica” are to a small degree overlapped with each other.

6.3 Density-based Clustering

DBSCAN from package *fpc* provides a density-based clustering for numeric data [Ester et al., 1996]. The idea of density-based clustering is to group objects into one cluster if they are connected to one another by densely populated area. There are two key parameters in DBSCAN:

- **eps**: Reachability Distance, which defines the size of neighborhood;
- **MinPts**: Reachability minimum no. of points.

If the number of points in the neighborhood of point α is no less than **MinPts**, then α is a *dense point*. All the points in its neighborhood are *density-reachable* from α and are put into the same cluster as α .

The strengthes of density-based clustering are that it can discover clusters with various shapes and sizes and is insensitive to noise. As a comparison, k-means tends to find clusters with sphere shape and with similar sizes.

```
> library(fpc)
```

by using `mclust`, invoked on its own or through another package, you accept the license agreement in the `mclust` LICENSE file and at <http://www.stat.washington.edu/mclust/license.txt>

```
> newiris <- iris[-5] # remove class tags
> ds <- dbscan(newiris, eps=0.42, MinPts=5)
> # compare clusters with original class labels
> table(ds$cluster, iris$Species)
```

	setosa	versicolor	virginica
0	2	10	17
1	48	0	0
2	0	37	0
3	0	3	33

In the above table, “1” to “3” in the first column are three discovered clusters, while “0” stands for noises, i.e., objects that are not assigned to any clusters. The noises are shown as black circles in figures below.

```
> plot(ds, newiris)
```

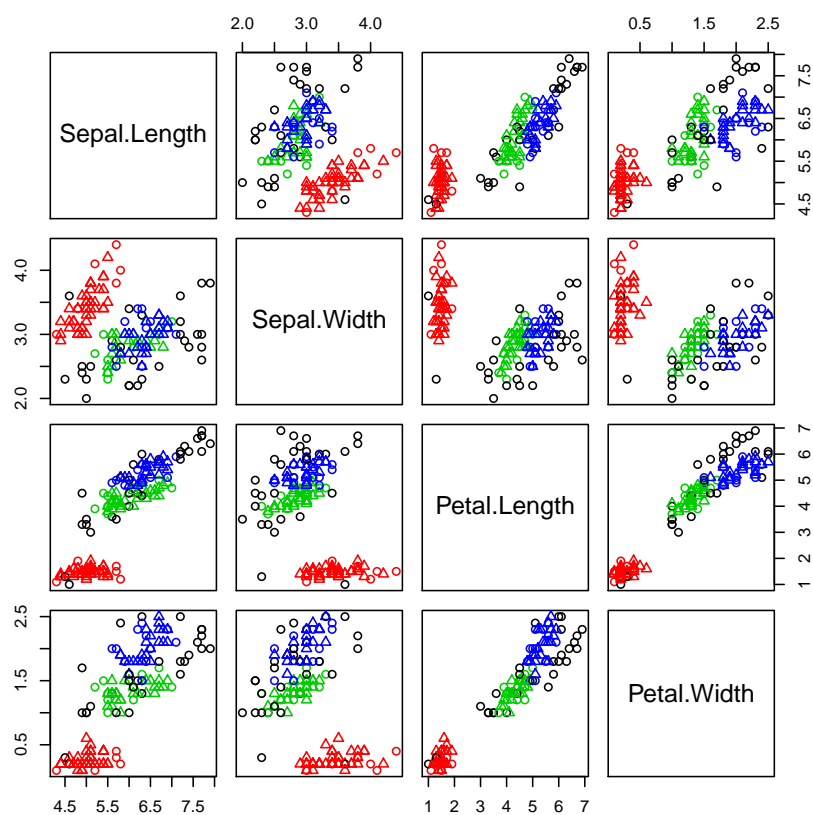


Figure 24: Density-based Clustering - I

```
> plot(ds, newiris[c(1,4)])
```

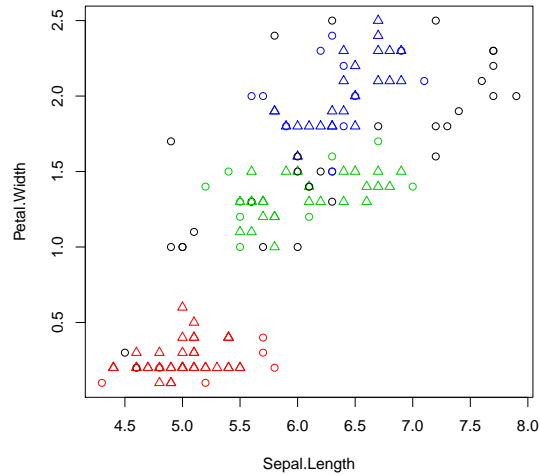


Figure 25: Density-based Clustering - II

Another way to show the clusters. Note that the data are projected to distinguish classes.

```
> plotcluster(newiris, ds$cluster)
```

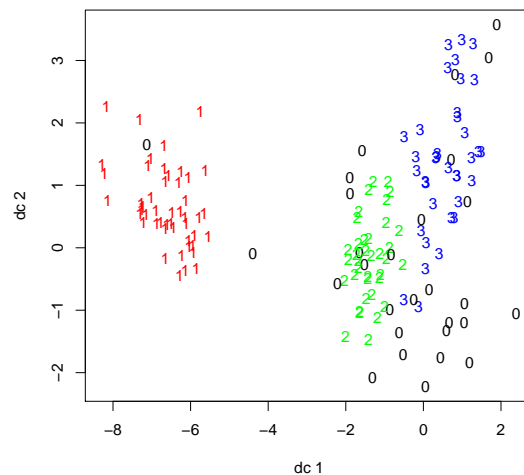


Figure 26: Density-based Clustering - III

Label new data

The clustering model can then be used to label new data, based on the similarity between a new object and the clusters. The following example draws a sample of 10 objects from iris and adds small noise to them to make a new dataset for labelling.

```

> set.seed(435)
> idx <- sample(1:nrow(iris), 10)
> newData <- iris[idx,-5]
> newData <- newData + matrix(runif(10*4, min=0, max=0.2), nrow=10, ncol=4)
> # label new data
> myPred <- predict(ds, newiris, newData)
> # check the labels assigned to new data
> plot(newiris[c(1,4)], col=1+ds$cluster)
> points(newData[c(1,4)], pch="*", col=1+myPred, cex=3)
> # check cluster labels
> table(myPred, iris$Species[idx])

```

```

myPred setosa versicolor virginica
0      0      0      1
1      3      0      0
2      0      3      0
3      0      1      2

```

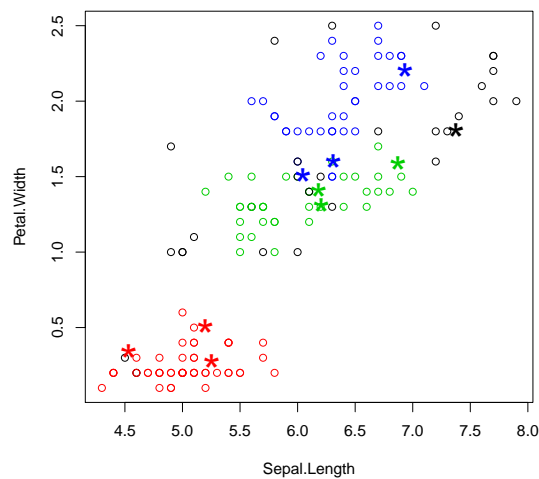


Figure 27: Prediction with Clustering Model

As we can see from the above result, in the 10 new unlabelled data, 8(=3+3+2) are assigned with correct class labels. The new data are shown as asterisk(“*”) in the above figure and the colors stand for cluster labels.

7 Outlier Detection

Package *extremevalues* []: Univariate outlier detection

Package *mvoutlier* []: Multivariate outlier detection based on robust methods

Package *outliers* []: Tests for outliers

This section is not available yet in this version.

8 Time Series Analysis

This section presents examples on time series decomposition and forecast.

8.1 Time Series Decomposition

Decompose a time series into seasonal, trend and irregular components using moving averages.

Data `AirPassengers`: Monthly totals of Box Jenkins international airline passengers, 1949 to 1960. It has $144(=12*12)$ values.

```
> apts <- ts(AirPassengers, frequency = 12)
> f <- decompose(apts)
> # seasonal figures
> f$figure

[1] -24.748737 -36.188131 -2.241162 -8.036616 -4.506313 35.402778
[7] 63.830808 62.823232 16.520202 -20.642677 -53.593434 -28.619949

> plot(f)
```

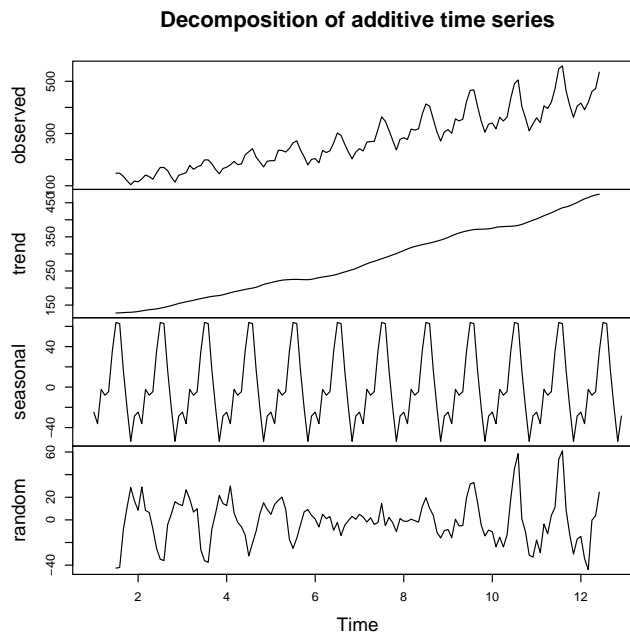


Figure 28: Time Series Decomposition

The first chart is the original time series. The second is trend in the data, the third shows seasonal factors, and the last chart is the remaining components after removing trend and seasonal factors. Some other functions for time series decomposition are `stl` and `decomp` from package *timsac*.

8.2 Time Series Forecast

Fit an ARIMA (autoregressive integrated moving average) model to a univariate time series, and use it for forecasting.

```

> fit <- arima(AirPassengers, order=c(1,0,0), list(order=c(2,1,0), period=12))
> fore <- predict(fit, n.ahead=24)
> # error bounds at 95% confidence level
> U <- fore$pred + 2*fore$se
> L <- fore$pred - 2*fore$se
> ts.plot(AirPassengers, fore$pred, U, L, col=c(1,2,4,4), lty = c(1,1,2,2))
> legend("topleft", c("Actual", "Forecast", "Error Bounds (95% Confidence)"),
+       col=c(1,2,4), lty=c(1,1,2))

```

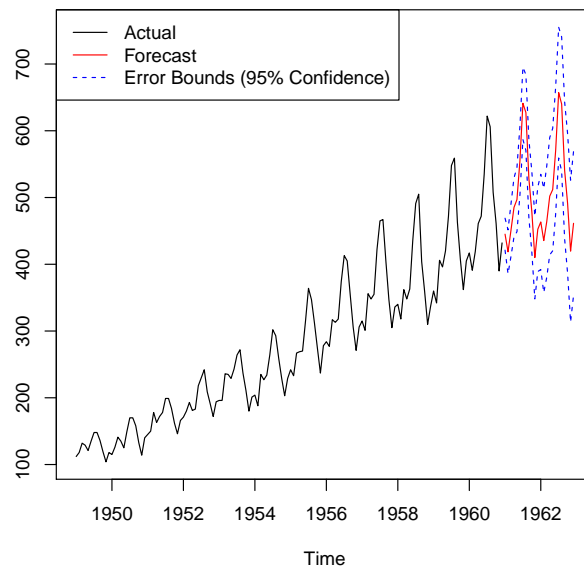


Figure 29: Time Series Forecast

The red solid line shows the forecasted values, and the blue dotted lines are error bounds at a confidence level of 95%.

9 Association Rules

Package *arules* []

Package *arulesNBMiner* []

This section is not available yet in this version.

10 Sequential Patterns

Package *arulesSequences* []

This section is not available yet in this version.

11 Text Mining

Package *tm* [1]: A framework for text mining applications within R.

Package *tm.plugin.mail* [2]: Text Mining E-Mail Plug-In. A plug-in for the *tm* text mining framework providing mail handling functionality.

Package *textcat* [3] provides N-Gram Based Text Categorization.

Introduction to the tm Package – Text Mining in R <http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf>

Text Mining Infrastructure in R <http://www.jstatsoft.org/v25/i05>

This section is not available yet in this version.

12 Free Online Resources

There are many free online resources on using R for data mining, and some of them are listed below.

- R Reference Card for Data Mining
<http://www.rdatamining.com/docs>
- Quick-R for SAS/SPSS/Stata Users
<http://www.statmethods.net/index.html>
- Data Mining with R - Learning by Case Studies
<http://www.liaad.up.pt/~ltorgo/DataMiningWithR/>
- Data Mining Algorithms In R
http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R
- Time Series Analysis and Its Applications: With R Examples
<http://www.stat.pitt.edu/stoffer/tsa2/>
- An Introduction to Recursive Partitioning Using the RPART Routines
<http://www.mayo.edu/hsr/techrpt/61.pdf>
- Data Mining Desktop Survival Guide
<http://www.togaware.com/datamining/survivor/>
- An R Time Series Tutorial
http://www.stat.pitt.edu/stoffer/tsa2/R_time_series_quick_fix.htm
- Time Series Analysis with R
http://www.stat.oek.wiso.uni-goettingen.de/veranstaltungen/zeitreihen/sommer03/ts_r_intro.pdf
- R Functions for Time Series Analysis
<http://cran.r-project.org/doc/contrib/Ricci-refcard-ts.pdf>
- Data Mining Algorithms In R
http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R
- The R Journal
<http://journal.r-project.org/current.html>
- R Tutorial
<http://www.cyclismo.org/tutorial/R/index.html>
- Text Mining Infrastructure in R
<http://www.jstatsoft.org/v25/i05>

References

- [Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231.
- [Frank and Asuncion, 2010] Frank, A. and Asuncion, A. (2010). UCI machine learning repository. university of california, irvine, school of information and computer sciences. <http://archive.ics.uci.edu/ml>.
- [Han and Kamber, 2000] Han, J. and Kamber, M. (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [Hand et al., 2001] Hand, D. J., Mannila, H., and Smyth, P. (2001). *Principles of Data Mining (Adaptive Computation and Machine Learning)*. The MIT Press.
- [R Development Core Team, 2010a] R Development Core Team (2010a). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.
- [R Development Core Team, 2010b] R Development Core Team (2010b). *R Data Import/Export*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-10-0.
- [R Development Core Team, 2010c] R Development Core Team (2010c). *R Language Definition*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-13-5.
- [Therneau et al., 2010] Therneau, T. M., Atkinson, B., and Ripley, B. (2010). *rpart: Recursive Partitioning*. R package version 3.1-46.
- [Venables et al., 2010] Venables, W. N., Smith, D. M., and R Development Core Team (2010). *An Introduction to R*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-12-7.
- [Witten and Frank, 2005] Witten, I. and Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, CA., USA, second edition.